Sced:
Constraint Based Scene Editing
User's Guide

Stephen Chenney

Basser Department Of Computer Science, University of Sydney

1. INTRODUCTION

Sced is a program for creating, visualizing and editing scene files for a variety of rendering programs. It runs in a UNIX and X windows environment. The aim was (and still is) to allow scenes to be designed rapidly, accurately and with little practical regard to the target rendering program or the machine it runs on.

Sced currently produces input files for the following programs:

- POVRay: A freeware raytracer for various platforms.
- Radiance: A freeware radiosity program.
- Rayshade: A freeware raytracer for UNIX platforms.
- RenderMan®: Pixar's standard file format for scene descriptions.
- VRML: The Virtual Reality Modeling Language - modeling on the World Wide Web.
- Genscan: A generic polygonal data file format.
- Genray: A fast and simple raytracer for UNIX platforms.

Sced may also be used to create OFF files, capable of describing polyhedral wireframe objects.

Sced has a secondary use as a wireframe viewing program. It supports a simple input file format for describing scenes, and is capable of reading OFF format files.

2. PRELIMINARIES

2.1 Regions Of The Window

The window consists of 5 distinct regions.

On the left is a sequence of buttons - the command buttons. Most of the program's functionality is invoked via these buttons. Some of the buttons are menu buttons - they pop up a menu when pushed. Such buttons have a small symbol to the left of the label, indicating a menu.

The largest window area is the *view window,* in which the scene appears. You may request a view window size larger than the main window size, in which case the view window will be contained within standard X Athena scrollbars.

In the bottom left corner of the main window is the *apply* button which is used in various situations, but most often to apply the text shown in the *text entry window* or to complete an operation.

The *prompt label* is beside the apply button at the bottom of the window, and generally gives some indication of what is currently displayed in the *text entry window*.

Finally, the *text entry window* runs across the bottom of the main window. It is used to obtain optional text where appropriate. It is a standard Athena text widget, with all the editing functions that go along with that. Note, however, that the return key has been mapped to the apply button, so hitting return while in this window is the same as clicking on the apply button. See Appendix A for a full list of the editing commands available in this window, or any other text entry area.

2.2 Command Button Functions

Command buttons are shown with or without a bitmap to the left of the label. Those with the bitmap (lots of little parallel lines) are menu buttons.

The following functions are available (listed according to the button they appear on.)

- **File:** Load, Merge, Save, Save As, Export, Export Displayed, Preferences, Copyright, Quit
- **CSG Window**
- **Wireframe:** Load, Delete
- **Object:** New, Edit, Copy, Delete, Name, Attribs, Alias, Dense Wire, Thin Wire, Change Base
- **Lights:** Point, Spot, Area, Directional, Ambient
- **View:** Viewpoint, Pan, Lookat, Lookup, Distance, Eye
- **Window:** Zoom, Image Size, Draw Mode, Save, Recall, Delete
- **Layers:** New, Add Objects, Merge, Display

- **Target**
- **Camera:** Edit values, Show/Hide, Edit Object
- **Preview**
- **Clear:** Clear Objects, Reset
- **Edit**

2.3 Selection

There are 2 forms of selection - object selection and point selection. Object selection is used to choose objects for editing etc. Point selection is used when a specific points are needed, such as for the specification of constraints.

2.3.1 Object Selection To select an object, drag over any of its edges while holding down button 1. To deselect, use button 2 or 3. A single mouse click is equivalent to dragging out a small rectangle centered on the pointer.

Selected objects are shown highlighted. They will remain selected until they are deselected or they become invisible for some reason (such as through deletion or a layer display change). Objects are generally NOT deselected by any of the functions that manipulate them.

2.3.2 Point Selection When the program needs a point to be entered, a small red square will appear on the nearest available point to the mouse. A mouse click **in** this square will select that point. Selected points appear in blue (by default). If there are no more points required, the scene will return to its previous state. Selecting a point a second time will deselect it. Selection doesn't happen until the button is released, so you may change your mind by moving the mouse off the point before releasing the button.

It matters which button you use to select a point. Button 1 will select the point by *reference, parameter or vertex*, button 2 will select as an *offset* and button 3 will select as an *absolute* point. Currently which button you use will only make a difference when selecting points for constraints, hence the full implications will be discussed in terms of the editing interface. Suffice to say here that button 1 will do for most purposes.

If it is valid to do so, a prompt will also appear in the text entry box at the bottom of the screen. You can type in a 3-vector here which nominates the corresponding point in the world. A vector is a sequence of three space separated real numbers in pretty much any useful number format (those accepted by sscanf). When you have typed in the vector (eg *1.0 -2.3 5.0*), either hit the return key or click in the *apply* button to enter the text. If an invalid sequence was entered the text will be ignored. If it is appropriate for the point to be an offset point, a dialog will appear allowing you to select the point type to use. If offset points are invalid in this context, the point is selected as an *absolute* point.

2.4 A Word on Text Entry

Some dialogs requiring text entry have the return key mapped to the *Done* or *Apply* button, so that a mouse click is not required. In general, such dialogs only require the entry of one piece of text. One example is the Ambient light dialog. A beep will sound if the return key is used at an inappropriate time for other dialogs. The text entry window at the bottom has the return key mapped to the apply button.

2.5 Changed Scene

Anything that changes the properties of an object, including its geometry and attributes, changes the scene as a whole. Also, functions that add or remove entities from the program environment, such as CSG objects, change the scene. Conversely, view changes do not change the scene, because they make no difference to the objects in the scene, only the view of them.

If the scene has changed, certain functions will ask for confirmation of some sort, generally whether you wish to save the current scene first. After a save or load operation the scene is always considered unchanged.

2.6 Cancel

Cancel buttons are provided for almost all dialogs. Choosing cancel will exit the dialog leaving the scene as unchanged as possible.

2.7 Colors

All the colors mentioned in this guide are the compiled in defaults. All colors may be changed through the appropriate X resource. See the section on X Resources for details.

3. BASIC FUNCTIONS

3.1 File Functions

3.1.1 Load If the scene has changed, you will be prompted to save first. Choose save to go to the save dialog, or load to get the load box.

In any case, the load dialog will eventually appear. The file name may be entered at the top, or can be selected from those shown in the windows below. The windows show the files in 3 directories. Each directory is a subdirectory to it's immediate left neighbor. To select a file, click on the filename. To view another directory, click on the directory name (shown with a / following its name). If there are more files than will fit in the window, a scrollbar is available. Click on Load or hit *return* to load the file shown at the top.

Any existing objects will be destroyed upon loading the file, as if the *Reset* command was invoked.

Load recognizes filenames ending in .Z, .gz or .z as compressed files, and automatically uncompresses them as it loads. External format files are also automatically recognized, and will be loaded appropriately. See the *External File Format* section for details.

3.1.2 Merge Merge adds the contents of a file to the scene currently being edited, without changing any existing features. It is most useful for loading predefined CSG objects for use in the current scene.

The dialog for selecting a file to merge is identical to that for load.

3.1.3 Save The file will be saved to the current file name. If no file name has been specified, the behavior is the same as *Save As*.

3.1.4 Save As A save dialog, similar to that for load, will appear. See load above for details. Enter a filename at the top, or select a file. You WILL NOT be warned about overwriting an existing file. Choose Save or hit *return* to save to the named file.

The scene is saved in a text format. Cameras, viewports, CSG trees and the like are all saved.

If the *Compress* option is in force, the actual name of the file saved may have an extension added, such as .gz. See the Defaults File section for more details.

3.1.5 Export The Export function exports all the instances in the scene as an input file for the target rendering program. If no target raytracer has been selected, you will be asked to select one. A dialog box, as for load and save, will then appear. The file produced is suitable for use as an input file to the specified raytracer. See the section **Renderer Specific Notes** for additional information.

A warning will be displayed if the camera has not been defined or there are no lights in the scene.

3.1.6 Export Displayed This behaves the same as Export, but only those instances and lights in displayed layers are exported. This allows control over the level of detail in the exported scene. It also allows you to prevent hidden objects from being exported.

3.1.7 Preferences This allows you to set various program options. These options are also accessible through the initial defaults file.

Autosave: Enter the number of minutes between automatic file saves. A time of 0 turns autosave off. Note that for large files the program will stop everything to save, which may have a large impact on interactive usage of the program. For smaller scenes the save time is not noticeable. If autosave is active and no file name has been specified, a dialog will pop up the first time the file is saved, asking for the filename.

Compress Output: Turns on or off compression of saved files.

Save Simplified Wireframes: Controls whether or not the simplified versions of CSG wireframes are saved. If not, they are regenerated when the file is loaded. Saving simplified wireframes makes the file larger, but reduces file load time. Only turn it of if file size is an issue for you.

Scene Path: The directory to look in for scene files.

Default Attributes: Pops up an attributes dialog box, from where you can set default object attributes.

Save: Saves the preferences to file. You are prompted for a filename. The default is a file called .scenerc in your home directory, or the defaults file specified on the command line.

3.1.8 Copyright A copyright message is displayed.

3.1.9 Quit If the scene has changed a save prompt will be issued, otherwise the program exits quietly.

3.2 CSG Window

The CSG Window is brought to the front (or created if necessary) and all the selected objects are transferred to that window for use there. Any objects removed in this way will be taken off the edit list. Edit constraint references will be maintained between objects that are selected, otherwise they will converted to offsets or absolutes. This prevents dependencies between objects in different windows, and ensures that CSG defining objects only have references within themselves.

Furthermore, you are prompted regarding the attributes attached to the object. You may elect to remove all attributes attached to the transferred objects (select *Yes*), or you may leave then defined (select *No*). Removing attributes just undefines them, it does not clear the values. You will generally want to clear attributes. For the reasons why, see the section on the *CSG Interface*.

For a complete description of the CSG object interface, see the later section *The CSG Interface*.

3.3 Wireframe

3.3.1 Load An ASCII OFF file, describing a wireframe object, may be loaded using this option. The standard load dialog is presented, allowing you to select a file to load. The file you load should be the header file for the object (NOT the geometry file). If the global environment variable OBJ_PATH is set it will be the default path for the files.

For a description of the OFF standard, see Randi J. Rost "OFF - A 3D Object File Format" for details. This document is available with the off distribution, a copy of which has been placed at ftp.cs.su.oz.au://stephen/off.tar.gz.

The object loaded is made available as another basic object class, just like a completed CSG object. They are accessed via the *Wireframe* button in the New Object dialog box.

Wireframe objects should not in general be used in CSG objects. Most of the example objects provided with the OFF distribution will cause errors if used as a component in a CSG object. However, wireframes satisfying the following properties may be used in CSG:

- Each polygon in the wireframe must be planar and convex.
- Polygons must intersect ONLY along complete edges or at single vertices.
- The object must be closed.

Note that the majority of objects in the off example database fail on point 1 (because of numerical inaccuracy) and on point 3.

3.3.2 Delete This button is only active if wireframe objects have been loaded. A dialog displaying the currently available wireframe objects is displayed. Select an object to delete. Objects that have instances cannot be deleted.

3.4 Object Functions

3.4.1 New Object A dialog is presented for selecting a new primitive object. The objects available are Cube, Sphere, Cylinder, Cone, Torus, Triangle, Plane and Patch. To select on object click on its picture. You can also type a number at the keyboard, to get that many of the object (eg hitting 3 while inside the cube image gives you 3 new cubes. 0 gives 10).

Selecting the *CSG Object* button within the New Object dialog will cause another dialog to be displayed, this time showing all the CSG objects currently defined. Use the scrollbars to view the range of objects available. Note that the CSG Object button will be insensitive if there are no CSG objects defined.

The *Wireframe* button displays the currently loaded wireframe objects, which may also be selected. This button is insensitive if no wireframe objects have been loaded.

The new object is placed at the origin of the world. It is also added to the list of objects awaiting editing. It will have a name somehow indicative of its type and the number of objects created so far. It has the default attributes. Note that if the World layer is not being displayed, then the object will not be visible.

3.4.2 Edit This command places all the currently selected objects onto the list of objects waiting to be edited. Editing in this context refers to the movement, scaling and rotating of an object. If only one object is selected, it is immediately edited. There is a shortcut for this operation — hold down the Ctrl key while dragging over the object. The object will be selected and edited.

3.4.3 Copy The selected objects are copied. A dialog is popped up with a field for the number of copies. The default is 1. You may also choose to copy the objects in 2 ways. *Individual* copies each object as a separate entity. The copies are constrained exactly the same as the original — including dependencies. *System* copies the selected objects as a system. Constraints between objects in the system are kept within the system. As an example, consider a cone constrained the sit on top of a cylinder. If both objects are copied using *Individual*, the cone copy will be constrained to the original cylinder. However, if *System* is used, the cone copy will be constrained to the cylinder copy. The return key performs an *individual* copy.

The name of the copy is derived from the name of the original, by appending a number. In all other ways the copy is the same as the original. Note that it appears immediately on top of the original, so may be difficult to see. The new object is placed on the *Edit* menu ready for editing.

3.4.4 Delete You are prompted before all the selected objects are deleted.

3.4.5 Name For each of the selected objects, you are prompted for a new name. The current name is provided as a default. Names are for your benefit only. They are used to refer to objects on the edit list, and are printed as comments in exported files, allowing you to find the actual instance in the raytrace input file. The program places no restrictions on uniqueness or any other aspect of names.

3.4.6 Attributes (Attribs) A dialog box for object attributes is presented. The defaults are those for the first object found on the list of selected objects. When completed, the attributes of all the objects on the list are changed. This is useful for changing the surface features of a large number of objects at once. If lights are among the objects selected, separate dialogs will be presented for them. See the section on Lights for details.

The simple surface attributes available are:

— *Color:* Enter normalized values ($0 \leq x \leq 1.0$) for red, green and blue components of the object's color. This is the color before any other effect is applied, ie all other surface effects modify this color.

— *Diffuse Reflection:* The diffuse component of an object's final color models how much light from a source it radiates back in all directions. A value between 0 and 1 should be entered here.

— *Specular Coefficient and Power:* These parameters are used to model the highlights that appear on an object's surface due to reflection of a light source. The coefficient determines how much of the incident light is reflected in this manner. The power determines how large the highlight will be - the smaller the power the larger but less bright the highlight. The coefficient should be a normalized value, the power can be anything positive.

— *Reflection Coefficient* This parameter controls how much light from other objects is reflected by the object. It is reflection in the sense of mirror reflection. Enter a normalized value, but you should be aware that reflective objects take much more time to render.

— *Refractive coefficient:* This is the coefficient of refraction for transparent objects. For instance air is 1.0, glass is 1.5. This parameter will only have an effect if the object has non-zero transparency.

— *Transparency:* This controls how much light passes through the object. A value of 1.0 means that the object is totally transparent, 0 means opaque. The light passing through is refracted according to the refractive index of the object. Like reflection, transparency dramatically increases the rendering time for an object.

It is also possible to set all the attributes to the default, by clicking on the *Default* button. All attributes on the selected bodies may be removed by choosing *None*. The latter is useful for controlling the attributes of components in CSG objects.

When all the attributes have been set, click on the *Done* button. The program attempts to parse all the values entered and set the attributes. If a given value could not be parsed the value of that particular attribute will remain unchanged.

In addition to the simple surface attributes, it is possible to attach an attribute string to the object. The meaning of this string and how it influences the rendered object depends on the renderer being used. To access such renderer specific attributes, click on the *Target Specific* button in the attributes dialog box.

3.4.6.1 Target Specific Attributes The *Target Specific* dialog box is not really target specific — it's the same for each target. However the information entered is specific to each target. The dialog has several features:

Texture String: Enter a string here to be exported along with the instance. For POVray it should be a complete, syntactically correct texture string. For Radiance, it should be the material name you want used for the object (a single word, no return). Rayshade requires a texture statement, which is exported immediately after the object. RenderMan requires one or more lines describing the shaders to use, including arguments to these shaders and always ending in a newline. VRML requires one of more valid VRML nodes, setting texture maps and the like. Genray and Genscan have no special attributes. The the *Renderer Specific Notes* for more information.

Transform Texture: This controls whether or not the texture is transformed with the object. If active, the same transformation that is applied to the object is applied to the texture (the texture is exported before the transformation for POV, or has a transformation applied in Rayshade). For RenderMan the shader is always exported after the transformation, so is always transformed. For Radiance this button has a different meaning — it causes the surface normals to be inverted.

Open: The Open toggle applies only to cylinders and cones. Cylinders and cones with this attribute set will appear without endcaps. Normally all cylinder and cones would appear with endcaps.

Declarations: A dialog is popped up allowing for any amount of text to be entered. This text is exported at the start of any file, so is the ideal place to put declarations, include files and set global variables. In fact, you can really do anything here. Any declarations will be in force for the rest of the file, so may be referred to by the Specific Attributes text. So, for example, if using Radiance you would declare all the materials here. For POVray you might #include the standard texture files. This string may be set from the Defaults file. See that section for details.

LOD: Only for VRML, this button provides access to the level of detail controls for an object. LOD works as follows:

- Only objects with dense wireframes have LOD.
- There are as many levels of detail available as there are less dense wireframes for the object. So if a cylinder has been made denser twice, it has 3 levels of detail (0, 1 and 2).
- The dialog displays the appropriate number of distances for the densest object selected. Fill in the change-over distances and select done.

Sensible things are done if more than one density level is present amongst the selected objects. In any case, it is probably wise to only do level of detail for objects of the same density at any one time.

Done: This button completes declaration of the specific attributes. These attributes are then used in preference to any simple attributes that may have been declared. Specific attributes are only used if the Done button in the Specific Attributes dialog is used. Using the Done button in the Simple Attributes dialog causes simple attributes to be reinstated.

Simplified: This button causes the Simple Attributes dialog to be presented again. In this situation any specific attributes will not be used, although they will be saved for later use. You may wish to use this method to disable specific attributes while previewing or test rendering.

Cancel leaves the specific attributes of an object unchanged, although if any simple attributes had been adjusted they will remain adjusted.

A separate Texture String is stored for each renderer. So you may enter a different specific texture for several renderers and export to each without adjusting attributes. Separate declarations are also kept for each renderer.

For Rayshade, RenderMan and VRML, the Simple Attributes are still used when exporting - generally to set the color or surface attributes. Hence it is always best to set Simple Attributes even if you will eventually use Specific Attributes.

3.4.7 Alias When an object is aliased, Sced will export something of your choosing rather than the object itself. So, for instance, you may alias a cube to a blob, and where the cube appears in the scene, the blob will be exported in it's place. This is most useful for exporting primitives that Sced doesn't support, or using objects that are defined somewhere else (and included in a renderer scene file via #include or some other mechanism).

When *Alias* is selected, a dialog is popped up allowing you to enter the string to be exported in the object's place. For Genray or Genscan the string is ignored. For all other renderers, the string will be exported in place of the object's usual type. A separate string is stored for each renderer and for each object.

The string should be syntactically correct for the target renderer. It should contain a valid object definition (although Sced doesn't check). To refer to the transformation applied to the object, you can use the special sequence of characters *(*transform*)*. In place of such a string, the current transformation will be exported. This has no meaning for RenderMan, for which the transformation is always exported immediately before the object. Nor does it do anything for Radiance, where the notion of transformation is hazy at best. Similarly, the special sequence *(*attributes*)* is replaced by the object attributes. This does not happen for RenderMan, where the attributes are exported before the object. Nor is the sequence replaced for Rayshade, because there is ambiguity about where to place the attribute specifiers (they go in 2 different places). VRML has the additional sequence *(*geometry*)* which is replaced with the object's geometric information.

Alias is the best means to introduce links into VRML scenes. Alias the object to the appropriate WWWAnchor or WWWInline node. Aliased objects are always surrounded by Separator nodes in VRML.

Be careful when using Alias with objects in CSG trees. The results may be very different to what you expect. The renderer may give an error, or nothing at all could happen (in Radiance).

3.4.8 Dense Wireframes The ruled objects (cylinders, cones, tori) and spheres have the option for dense wireframes, which use progressively twice as many generating lines, giving a much better approximation to the true surface. Use this command to convert selected objects to the next level of wireframe density. This option is essential for good results using Radiance and VRML. See the "Renderer Specific Notes" section for details.

3.4.9 Thin Wireframes This is the inverse function to *Dense Wireframe*. It makes a cylinder cone, torus or sphere wireframe one level less dense. You cannot reduce below the initial density (using 8 generators).

3.4.10 Change Base The *Change Base* function allows the base type of an object instance or instances to be changed. For example, you can change a cube to a cylinder. The function is most useful for adjusting the base type of CSG objects after modification of the original CSG type.

When the function is selected, the *new Object* dialog is presented. Select the new base type as if choosing a new object. When a new type is chosen, all the selected objects are adjusted to be of this new type. All that changes is the type. The same transformation is applied, the same constraints are active and so on. The only visible change is a switch in wireframes. Note that upon editing the Origin and Scaling point may appear to be in strange places not connected to any vertex of the object, but this causes no problem for the interface.

3.5 Light Manipulation Functions

3.5.1 Point A point light source is created at the origin. Its default intensity is 1.0 in all colors. The light may be selected and editing like any other object. Note that rotating and scaling a point light source has no meaning in the final scene, although it may help editing.

The dialog to change a point light's intensity is accessed through the normal Object Attributes menu. It asks for three values representing RGB intensities. If only one value is entered it is taken as the value for all three of red, green and blue. For most renderers the intensity should be between 0 and 1. However Radiance and RenderMan use point light sources with inverse square intensity, so higher values (much higher) are generally required.

3.5.2 Spot A Spotlight light source is created. Initially the spotlight is located at the origin, pointing in the negative Z direction. Editing the object controls both its position and the shape and direction of the "cone" of light emitted. The wireframe used is indicative of this cone.

The dialog to change a spotlight's parameters is accessed through the normal Object Attributes menu. Aside from RGB intensity values, it asks for several other things:

Outer Radius controls the falloff of light at the edge of the cone. The light intensity will begin to decline at the edge of the cone indicated by the object wireframe, with a apex angle of *a* degrees. It will have completely fallen to 0 by *Outer Radius* times this *apex* angle. So setting a value of 1 for *Outer Radius* means the light falls off immediately. A value of 1.5 indicates that the light takes some time to fall off. Note that values must always be greater than 1.

Tightness controls how quickly the light falls off at the edge. Higher values give sharper falloffs. The value should be between 1 and 100. This parameter affects the fuzziness of edges.

Invert causes the cone of light to be inverted. that is, the area within the cone shown on screen will be dark, and the area outside will be lit. This is how spotlight angles greater than 90 degrees are specified.

Note that Genray and Genscan do not have Spotlight primitives.

3.5.3 Area An area light source is created. Initially it is located at the origin, with the area lying in the X-Y plane. The object may be edited to move it and resize and reorient the area. The square wireframe used indicates the size and location of the light.

The dialog to change an arealight's parameters is accessed through the normal Object Attributes menu. Aside from RGB intensity values, it asks for several other things:

Xnum are the number of light sources in the X direction (as the light is drawn initially). This attribute only has meaning for POV and Rayshade, where an arealight is approximated by lots of point lights. More lights give a better result, but significantly increase rendering time.

Ynum is the number of light sources in the Y direction.

Jitter controls whether the point lights are jittered to give a better result. Jittering increases rendering time, but produces softer shadows and reduces banding in intensity.

Note that Genray and Genscan do not have Arealight primitives.

3.5.4 Directional A directional light source is created. Such a source emits light in a certain direction at a constant intensity. POVray, Radiance, Genray and Genscan do not support Directional lights.

The wireframe for a directional light indicates the direction. It is shaped sort of like a small pyramid. The direction is *from* the apex *to* the base (the direction of the default major axis). Note that the location and size of the object makes no difference — all that matters is the direction of that centerline.

3.5.5 Ambient You are prompted for the ambient light level in the scene. The light color should be entered as normalized RGB values. This light source will diffusely light all object in the scene, regardless of whether or not they are in shadow. Note that POVRay is unable to make full use of this feature, as it only allows for monochromatic ambient light. If only one value is entered, the program will assume it is an intensity value, and set each of the RGB values equally to that entered, giving white light.

3.6 View Manipulation Functions

A range of functions are provided for manipulating your view of the scene. There are several key viewing parameters:

- *Viewpoint:* is a vector from the center of the world out to the eye. In other words it controls which direction you are looking from.
- *Look At:* is a point in the world which is the center of the world for viewing purposes. This point will always appear in the center of the viewing window.
- *Look Up:* is a vector defining which way is up. A line, parallel to this vector and passing through the *Look At* point will always appear to be vertical on the screen, although it may be leaning into or out of the screen.
- *Distance:* refers to the distance from the *Look At* point to the abstract viewplane window you are looking through. Points lying behind the viewplane will have strange things happen to them, and most likely become invisible. This distance also influences the apparent size of the scene.
- *Eye Distance:* is the distance from the viewplane to the eye. This controls the amount of perspective in the scene - shorter eye distances relative to viewplane distances give more perspective. For parallel views, set this value to be something

large relative to the viewplane distance. Eye distance also influences the apparent size of the scene - shorter eye distances make the scene appear smaller.

3.6.1 Viewpoint The program goes into change view mode, where most buttons are desensitized and the cursor is a cross. The cursor is theoretically lying on a sphere, centered on the *Look At* point with poles in the *Look Up* direction. The current viewpoint is a point on this sphere. Moving the mouse moves the viewpoint on the sphere in the following way:

— *Button 1 Motion:* The viewpoint is free to move to any point on the sphere. Moving the mouse up or down rolls the world in the corresponding way, while moving sideways rotates the world around the *Look Up* vector. Moving the mouse all the way to the top of the window will give the impression of looking straight up the *Look Up* vector, while moving all the way to the bottom will have you looking straight down it. Moving from one side of the window to the other will do a complete circuit about the world. You may find it easier ignore this description and learn by experimenting with the interface.

— *Button 2 Motion* Moving the mouse with button 2 down only allows for circling the world, it prevents moving the view higher or lower in the the scene.

— *Button 3 Motion* This allows only for apparent motion up or down in the scene.

The current viewpoint is shown in the text entry window. You can specify a particular viewpoint (in vector format) here. If for some reason your text is unacceptable, the view will remain the same.

Click on the *Apply* button to complete a viewpoint changing session. Clicking on *Undo* in the edit interface will also complete the view change.

Holding down the Shift key and dragging the mouse provides a shortcut for view changing which is available at any time (even if the menu is not active). The mouse buttons have the same effect as for a general view change. The view change lasts only during the mouse drag — it is automatically started and finished as the mouse button is pressed and released.

3.6.2 Pan This is like panning a film camera. It rotates the view around the current eye position, so the scene appears to slide past you. In other words, both the viewpoint and look at point are changing while the eye stays in the same position in the world. The mouse buttons control constrained motion similar to a viewpoint change.

The shortcut for a Pan operation is the Mate or Alt key. Holding down the Meta or Alt key and dragging the mouse is a single Pan operation available at any time.

3.6.3 Look At When this item is selected the program goes into *point selection* mode, where you may enter a new point for the center of the world. You can type in an arbitrary point if you wish. If the point entered in the text entry window is an offset point, the Look At point will be offset from its current location by the amount entered.

3.6.4 Look Up The program goes into *point selection* mode waiting for 2 points defining the new up direction. The *Look Up* vector will be the one *from* the first point entered *to* the second point. The first point may be de-selected by selecting it again. Each point can be textually entered at the bottom of the screen. The vector just chosen may not necessarily end up being vertical, due to perspective effects, but a vector parallel in the world, passing through the *Look At* point, will be.

3.6.5 Distance As with viewpoint changing, most of the buttons are desensitized. The cursor changes to an arrow. Button down mouse motion will cause the distance to change. Pushing up moves you closer to the scene, while dragging down moves you further away. Each button has a different speed - button 1 is medium, button 2 fast and 3 slow. You may also enter a new distance in the text entry window. The apply button completes a session.

3.6.6 Eye Distance Similar to distance, the eye distance is adjusted by pushing or dragging the mouse. Note that moving the eye closer will appear to make the scene smaller due to perspective effects. Again you may enter a value and the apply button completes a session. Eye distance is synonymous with the focal length parameter taken by many renderers.

3.7 Window Manipulation Functions

Window functions act with the View functions to control the current view of the scene.

— *Zoom:* controls the magnification of the scene.

— *Image Size:* controls the size of the window in which the scene appears. This is the window contained within scrollbars. The command does not adjust the outer window size. Note that it is the image size values that are sent to the renderer to control the rendered size of the image (although POV ignores this option). It is also the size used in camera calculations. If no image size has been specified, it defaults to the current size of the view window.

— *Draw Mode:* controls how back face edges are drawn.

— *Save:* Saves the current viewport for later recall.

— *Recall:* Recalls a viewport.

— *Delete:* Deletes a saved viewport.

3.7.1 Zoom (Magnification) Function A dialog appears asking for the new zoom value. This controls the apparent size of the image on screen. Clicking on the *Apply* button applies the new zoom without closing the dialog, whereas *Done* applies the value and closes the dialog. *To Fit* causes the zoom value to adjust to the largest value such that all the objects are visible in the scene. This is mostly useful for finding "lost" objects or expanding the scene to fill the available space. The return key is mapped to the Done button.

The + and - keys act as shortcuts for the Zoom function. + doubles the magnification. - halves it. These shortcuts are available at any time.

3.7.2 Image Size A dialog box appears asking for new width and height values for the *view window*. Enter positive integers for each value, and choose *Done.* Note that if the image size you enter is smaller than the available area in the program window, you will get a window larger than you asked for. However when the file is exported the value will be as you defined it. If you want to see how big it actually is, resize the program window until scrollbars appear around the view window.

The *To Fit* button sets the values to the current visible area, so there will be no scrollbars.

Cancel exits the dialog leaving the actual values unchanged, regardless of whether any of the values in the dialog have been modified.

3.7.3 Draw Mode There are three ways in which Sced draws back facing edges object edges. they may be drawn fully, dashed or not at all. The *Draw Mode* item pops up a dialog from which you may select the mode you wish to use. The current mode is shown selected as a default.

3.7.4 Save You are prompted for a name for the current view, which is then saved to be available via the Recall menu item. All aspects of the view are saved, including image size and zoom.

3.7.5 Recall A list of currently defined views is popped up. Choose the view you want by clicking on it. The stored view will be reinstated. Those views that appear in the defaults file will appear, as will any you have saved. One special view, called *Camera*, sets the view to match the currently defined camera, if it exists. Any views defined in the *Defaults File* will appear here also.

3.7.6 Delete A dialog appears with all the currently defined views. Choose the one you wish to delete and it will be removed from the list.

3.8 Layer Functions

The Layer functions provide control over which objects are visible on the screen. Conceptually, each object belongs to a layer. New objects belong to the *World* layer, unless they were copied or they have explicitly been added to another layer. At any given time, a set of layers is displayed, and those objects which appear in the displayed layers are visible. Objects in layers not displayed are not visible. Invisible objects are still exported and saved. An object can only belong to one layer at a time.

There are 2 special layers — World and Construction. The World layer is the default layer for new objects and lights. The Construction layer is never exported. Objects in the construction layer may be used just like the construction lines of geometry. They are displayed in green. Why are they there? There are some situations in which it is impossible to constrain objects in the way you want — generally because of cycles or self referencing restrictions. Construction objects provide a way around this. They are also very useful to place a vertex where none exists - such as at the midpoint of 2 other vertices.

3.8.1 New A new layer is created containing the selected objects. You are prompted for a name (a default is provided). The objects are removed from their previous layer, and the new layer is displayed.

3.8.2 Add Objects You are asked to select a layer, to which the selected objects will be added. The objects just transferred take on the visibility of the destination layer. If the new layer is not visible, they will also be removed from the edit or selection lists.

3.8.3 Merge You must select two layers. The first chosen is deleted and all the objects from that layer added to the second selected. If the *World* layer is one chosen, the other will be deleted regardless of the order selected. The objects transferred take on the visibility of the destination layer. If the new layer is not visible, they will also be removed from the edit or selection lists.

3.8.4 Display A popup containing all the defined layers is presented. Those displayed are shown highlighted. Each label is a toggle controlling the display of that layer. When a layer is toggled, the scene is redrawn immediately to reflect the change. Choose *Finish* when you are satisfied. Objects in layers that are hidden will be removed from the selection and edit lists.

3.9 Target

This button allows for the choice of the target renderer. This effectively only determines the format of the exported file, although it also determines which attribute strings are used and may effect some default values.

3.10 Camera

Sced offers a generic camera with the following parameters:

Location: This is the location of the eye of the camera (not the viewplane).

Look At: This point will appear in the center of the image.

Up: Controls which way is up.

Horiz and Vert FOV: Horizontal and vertical field of view. This controls how much of the scene is visible. Anything falling inside a pyramid with apex at Location and vertical angles horiz and vert FOV will be visible. These values also control perspective effects. They are only used if the *Use FOV* toggle is set.

Horiz and Vert Dist: Horizontal and vertical window sizes. These are the size of the view window into the world. They are redundant with FOV, but POV uses them so they appear here. These fields will only be used if the *Use FOV* toggle is not set.

Focal controls the distance from the location (eye) to the view plane. It is only required by those renderers that clip to the viewplane. Actually, it is completely redundant given the other parameters.

Use FOV: Only one of FOV and Dist are required to specify the camera. This toggle controls which is used. The unused fields will have no effect on the camera.

Implicitly the camera also relies on the Image Size and Zoom factor.

There are 3 options for manipulating the Camera:

3.10.1 Edit Values A dialog for the entry of camera data is presented. You can modify any parameter — each corresponds to a camera definition element.

Alternatively, you can use the *Viewport* button to match the camera to the current viewing specifications. This is the easiest way to define a camera.

When all is finished, choose the *Done* button, which will save the camera as defined. *Cancel* will close the dialog leaving the camera unchanged from its previous state.

3.10.2 Show/Hide This options causes the *Camera Object* to be displayed or hidden. The object shows the viewing pyramid, with the apex at the eye and the viewing window at the base. the extra line on one side indicates the top of the window. It is shown in purple by default. Displaying the camera allows its relative location to be seen. this is most useful if trying to place a camera inside a room or other confined space.

3.10.3 Edit Object The Camera Object is edited, just like any other object. When the editing session is completed, new values for the camera parameters are derived in the following way:

Location is taken from the location of the apex of the pyramid.

Look At is derived from the central axis of the pyramid (that is, a line joining the apex and the center of the base).

Look Up is in the direction of the line joining the center of the base to the extra line on the top side of the pyramid.

FOV parameters are taken as the angles subtended at the apex by the sides of the pyramid.

Focal distance is the length of the line joining the apex to the center of the base.

When deriving these values, one major thing is assumed: the object is still a pyramid — or the base is still a rectangle. Transforming the object such that the base is no longer a rectangle will have strange results.

The Camera Object may be constrained just like any other object, which I'm sure is useful in some way (generally for setting which way it points).

3.11 Preview

A preview dialog box is popped up, allowing you to specify the parameters for the preview. The renderer options are shown at the top. Choose the renderer you wish to preview with. You may also select a width and height for the preview image. The default is the size of the current image window. Since only the window size is adjusted using these parameters (not the camera), it is good to keep them in proportion to the current image size. The *Preview All* button causes all the objects to be previewed, rather than the default of just the selected objects. Choose *Go!* to proceed with the preview.

Setting the *Camera* toggle will cause the defined camera to be used rather than the current view (the default).

If no lights have been defined, a light located near the eye is used, otherwise the defined lights are used.

The selected objects are exported to a temporary file then previewed using a renderer chosen for the task. If the process fork succeeded a dialog will pop up indicating the command used for the preview, and asking for recognition. If for some reason the preview file cannot be exported, or the renderer cannot be forked, a message will inform you.

The target renderer is not necessarily used because it may be unsuited to previewing (Rayshade does not give a runtime view of the picture being traced and POVRay is slow but gives a dynamic color view.) The full pathname for each of the renderers needs to be defined in the defaults file, or compiled in. It is also possible to specify options for the preview within the defaults file or at compile time.

3.12 Clear

There are two variations of this function.

Clear Objects deletes all the objects in the Scene and CSG window. It clears any edit and selection lists also. It does not delete existing base objects.

Reset performs a *Clear Objects*, and then deletes all the base objects that have been defined - that is any CSG and Wireframe objects. It also deletes any saved viewports and layers.

4. EDIT INTERFACE

4.1 Overview

Sced is a *Constraint Based* editor, in that objects are manipulated by placing constraints on their location, shape and orientation. Constraints are also used to remove the 3rd dimension from the world for interaction, to allow meaningful interpretation of a 2D mouse point as a 3D world point. Constraints may also be *maintained* which means that the relative position of objects may remain the same even when one of the objects is moved.

When using Sced, object are controlled through 3 or 4 key points, or *features*. The location or orientation of each feature controls how the transformation applied to the object. The features are:

— *The Origin* controls the *position* of the object. The Origin appears in Green somewhere in the scene, initially at the center of the object. Moving the Origin point moves the body with it, and constraining it constrains where the body is in space. The Origin also performs 2 other key functions. It the the center for all rotation of the body. That is, the Origin will stay in the same spot as the body rotates around it. This lets you rotate about any point you like. The Origin is also the center for scaling the object. That is, the Origin stays where it is, regardless of how you scale the object. This means you can move the object while scaling, or scale about one corner of the object. The short, the Origin stays where you put it, no matter what other operations you apply. However you may choose an arbitrary location for the Origin with respect to the body. The Origin is the link between the object's location and the rest of the world.

— *Axes* control the orientation or *alignment* of the object. They appear as red, green and blue lines sticking out of the origin. The red, or *Major* axis is longer than the green, or *Minor* axis. The blue, *Other* axis is the shortest. The Major axis generally takes precedence over the Minor axis, and the Other doesn't control anything much. Scaling is along the axes. When the axes rotate, the body rotates with them. An axis may be constrained to be parallel to a line, or perpendicular to a plane.

— *The Scaling Point* controls *scaling* of the object. It appears in Red on one vertex of the body. When this point moves the object scales to keep the point at the same vertex. Scaling is about the Origin and along the Axes.

— For tori only, the *Radius Point* controls the major radius of the torus (the distance from the center of the torus to the middle of the tube). As the radius point moves the torus changes shape so that the distance from the Radius Point to the center matches the distance from a corresponding initial vertex on the object to the center. It's more intuitive than it sounds.

— For triangle and bezier patch primitives, there is one additional feature for every control point (the 3 vertices of a triangle, and 16 control points of a patch). These points are directly tied to the corresponding control vertex.

The cursor will change depending on what feature the mouse is over. If the feature under the mouse is not open to interaction, the cursor will be a closed circle. The cursor will be a pair of arrows in the rotation region. It will be a diamond cross in the origin region and a sizing arrow in the scale point.

The features just described may be constrained to satisfy geometric conditions. More specifically, the point type features (eg Origin point) may be constrained to satisfy one of the following conditions:

— *Laying in a Plane:* The point is constrained to lie in a plane. A plane is defined by a normal vector and a point in the plane. When constrained to a plane, the point has 2 degrees of freedom. it.

— *Laying on a Line:* The point lies on a line as defined by a direction vector and a point. The point retains 1 degree of freedom (along the line).

— *Laying on a Sphere:* The point must lie on the surface of a sphere. The sphere has a center and radius. Spheres control distances — the distance of the constrained point from the center of the sphere.

— *Laying on a Circle:* The point must lie on a circle. The circle has a center, radius and plane.

— *Located at a Point:* The point must be located at the constraint point. The Origin, Scaling or Radius point has no more freedom.

Axes may be constrained to direction vectors only. The axis must be parallel to the constraint vector.

The constraints themselves may depend on other objects in the scene. For instance, a plane constraint may be defined to be one face of a cube. As the cube moves, the constraint changes, and this in turn impacts on the initial object. In this way constraints can propagate through a scene — this is what is referred to as *constraint maintenance*. Carefully defined constraints, together with the automatic constraint maintenance system, make it possible to build scenes which require very little explicit modification to fix errors or make major visual adjustments. Furthermore, accuracy is easy to attain. It is a simple matter to align objects or accurately size them. The accuracy is maintained too, so one need never explicitly align objects twice.

The picture so far is of a set of points which satisfy a set of conditions and subsequently cause the object to behave in a certain way. But how are the constraints specified?

4.2 Specifying Constraints

Placing constraints on an object is as simple as selecting the desired constraints off a list. Each feature has a list of constraints associated with it. These constraints are referred to as the *Available Constraints* for the feature. The lists appear in the Edit dialog box (which appears when an object is edited). Each constraint has a name associated with it, and to make the constraint *active* you simply click on the name. To make a constraint *inactive*, click again. Active constraints are shown with their buttons inverted (like toggles). Inactive constraints look like normal buttons.

When a constraint is activated, the system ensures that the relevant feature satisfies the constraint. The exact process for ensuring constraint satisfaction is explained below.

More than one constraint may be active for a feature at any given time. For instance, an object may be constrained to both a sphere and a plane at the same time. In this situation the constraints are combined to give a single resultant constraint. For the example just given, the resultant is a circle — the geometric intersection of a plane and a sphere.

If a feature has too many constraints imposed upon it, the constraints may be inconsistent. That means there is no way that the feature can satisfy all the constraints at once. When such a case arises, the feature will not be adjusted in any way.

Conversely, a feature may be under-constrained. That is, there are too few constraints on the object to entirely define it's location. The object is said to have some degree of freedom. In this case you are able to interactively manipulate the feature within the bounds of the imposed constraint. See the section on *Interactive Manipulation* for details.

4.2.1 Default Constraints There are several default constraints provided for each feature. The defaults are intended to provide general purpose constraints for looks-right manipulation. In these situations speed and simplicity is most important. Accuracy is not an issue.

For the Origin there are 3 default constraints. Each one is a plane, corresponding to the world axis planes. Hence there is a constraint that passes through the Origin and has a normal parallel to the X axis. There is a corresponding constraint for the Y and Z axes.

The Scaling Point has, in general, 4 defaults. The first 3 are planes passing through the Scaling Point with normals parallel to the Major, Minor and Other Axes. In addition, there is one more constraint which depends on the particular object. This may be a constraint to support uniform scaling, or scaling to change the radius of a cylinder.

Each axis has a default that is the current axis. These defaults are intended only for interaction purposes.

The Radius point has a single default, which is a line constraint allowing simple adjustments to the radius.

4.2.2 Adding New Constraints New constraints are added to the list of available constraints for a feature by selecting the desired constraint type from the *Add* menu under the constraint list for that feature. That's more intuitive than it sounds. The menu contains one entry for each of the constraint types suitable for that feature.

When a constraint type has been chosen, a dialog pops up specific to that type. The dialog has spaces for all the information required to specify the constraint. Some slots are optional. Other slots have more than one possible label. These have menus to the left of the slot. Choose the appropriate label off the menu. The possible labels for each constraint type are listed below, along with a summary of how the slots combine to define the constraint.

Plane:

- *Point*: This is a simple point that will lie in the defined plane.

- *Normal*: A normal point will in some way define a normal.

To define a plane, the first three slots must be filled. The third slot may also be filled. The combinations of Points (Plane Pts) and Normals (Normal Pts) and the resulting plane are as follows:

3 Points: The plane defined is that which passes through the 3 points given.

2 Points and a Normal: The plane is that which contains the line joining the 2 points and is perpendicular to the plane containing the 2 points and the normal. Draw a picture - it helps. Or experiment.

1 Point and 2 Normal Pts: The plane is that which passes through the point and is normal to the line joining the 2 normal points. So the normal points define the plane normal and the first point says where it is.

1 Point and 3 Normal Pts: The plane is that passing through the first point and parallel to the plane passing through the 3 Normal Pts.

The easiest way to define a plane is via the first method. Simply select 3 points on the plane. This is the default method (that of least resistance).

Line: There are 2 valid combinations of filled slots.

2 Points: The line defined is that passing through the 2 points.

2 Points and a Normal Pt: The line defined passes through the first point and is parallel to the line joining the second and third points.

The easiest way to define a line is by specifying just the first 2 points, to give a line joining those points. The Normal Pt is optional.

Point: Only one possibility here. The point entered is the constraint point.

Sphere: To define a sphere the Center slot and at least one Radius slot must be filled. If only one Radius slot is filled, the sphere defined is that with the entered Center and Radius the distance from the Radius point to the Center (that is, the Radius Pt lies on the sphere). If 2 Radius slots are filled, the radius of the defined sphere is the distance from one radius point to the other. This allows the radius of the sphere constraint to be "measured off" from some other distance in the scene.

Circle: A circle is a plane combined with a sphere. So both a plane and a radius must be specified. The labels are the same as those for a Plane and Sphere combined, and the combinations work exactly the same, with the minor exception that no Radius Pt need be specified, it defaults to the second point entered. The first Point is the center of the circle.

Ratio Point: A ratio point constraint is a point constraint that is defined in a different way. The first 2 slots must be filled. These are the points that the ratio is relative to. The third slot is for the ratio itself. It defaults to 1:1, or the midpoint of the 2 Points given. Any ratio may be specified, in the form *DistToPt1:DistToPt2*. The defined point is located so that its distances to the points are in the given ration. Negative distances may be used to give external ratios (the point will not lie between the two points given, rather somewhere on the line beyond them).

Ratio Plane: A ratio plane constraint is a plane that is defined by a normal joining 2 given points and a location given by its distance ratios from those 2 points. As for the Ratio Point constraint, 2 points must be given. The plane will be normal to the line joining the 2 points, and it will pass through the corresponding ratio point. Once again the Ratio itself is optional — it defaults to 1:1, or the midplane.

Axis: An axis is a direction — a line without location. There are 3 Points, at least 2 of which must be filled. If 2 slots are filled, the axis points in the same direction as the line joining the 2 points. If 3 slots are filled, the axis is normal to the plane defined by the 3 points.

To fill a slot, the point must somehow be selected. This is the easy part. When the Dialog pops up, the window goes into point selection mode. Note also the little arrow toggles beside each slot. Only one slot will have an inverted at any time. That is the slot that will be filled when a point is selected, or the *Destination* slot. The destination slot may be changed simply by clicking on the button beside the desired slot.

The point may be selected by clicking on the required vertex in the window, or by entering explicit values in the text window below the view window. The destination slot is filled with the selected point and the destination is automatically reset to the next required slot. If you change your mind about which point you wanted, you can reset the destination slot and enter the point again.

WHICH MOUSE BUTTON YOU USE MAKES A DIFFERENCE. The mouse button used in the selection determines what type of point is selected. The type of point used to define a constraint determines how the constraint is maintained.

Reference Points (button 1) are attached to a particular object. As the object they are attached to moves, so does the defining point. Use this type of point to specify relationships between objects that you want maintained. When choosing a reference point, button 1 down will cause the object referenced to change color. If this is not the object desired, move the mouse with the button down to cycle through other objects which may be referenced by that point. Button up inside the point rectangle chooses the currently highlighted object.

Bezier Patch and Triangle objects have slightly different behavior for points selected with button 1. If a control point is selected, the specified constraint point moves with the control point. If a point on a bezier patch is selected, the point is attached to a certain position on the patch in parametric space. In general that means that it behaves the way you want.

Offset Points (button 2) stay in the same place relative to the current object. Use these to specify scaling constraints which relate to the body being edited only. If offset points are entered via the text entry window, the vector entered is taken as the offset from the center of the body. Offset points are also useful as origin line constraint specifiers, as they encode motion in a particular direction.

Absolute Points have a fixed position in space. Use them to position an object somewhere where you don't want it to move from.

Reference points are the basis for constraint maintenance. They allow the current object to react to changes in the object it references. There are limitations on what can be used as a reference point. In particular, reference points cannot lead to cycles. The system will not allow references to objects that depend on the current object. So you can't say object A is the same height as object B, and then say that object B is the same height as A. If a point is selected that violates this rule, the system will change it's type and inform you of the change. Often this is an indication that you chose a different point to the one intended, so you may wish to try again (by reseting the destination slot and selecting again).

Constraint Maintenance, and how the various point types impact upon it, is explained in more detail shortly.

An alternative way to select some points is by clicking on the button may appear to the far right of the slot. It may be labeled Origin, Scaling or something else. If there is no label on the button, it will have no effect. Clicking on the button (if it is labeled) sets the slot to be the same as the feature named by the label. So clicking on the Origin button beside a slot sets the slot to the location of the current object origin. Furthermore, the point will always match the Origin, even if the Origin is redefined to be somewhere else. Often the use of a feature point allows you to avoid having object constraints depending on the object itself, which is not normally allowed.

When a point is selected, it remains highlighted in the window, providing feedback as to which points have been selected.

While selecting points, all the view changing shortcut keys are available, so you may change the Viewpoint, Pan or Zoom.

There are several other buttons or fields common to every Add dialog.

Label Field: To the left below all the slots is a text field containing a label for the constraint. This is the label that will appear in the constraint list when the constraint is added. A default label is provided, but you are free to change it to whatever you like. It is a good idea to keep it short.

Selected Toggle: If this toggle is active, the constraint will be added in an active state. This is simple a shortcut to adding the constraint and then selecting it to make it active. Most often you will want to add a constraint selected, otherwise why would you be defining it.

Done Button: This button is insensitive until the minimum number of slots have been filled. As soon as sufficient slots are filled, it becomes available. Selecting this button adds the constraint to the list of available constraints.

Reset Button: Reset clears all the slots, allowing you to start again.

Cancel Button: This cancels the process at any stage.
In addition there is a *Remove* button that will always be inactive when adding a constraint.

4.2.3 Modifying Existing Constraints An existing constraint may be modified by clicking on the *Modify* button for the required constraint list, then clicking on the constraint you wish to modify in the list. Clicking on anything else should cancel the operation. Constraints are deleted (removed) in this manner too (modified out of existence).

When a constraint is selected for modification the Add dialog pops up with all the slots filled as they were when the constraint was created. You may then proceed as you would if you were defining the constraint for the first time.

If the constraint is not active, the *Remove* button is available to delete the constraint entirely. Cancel cancels the process, leaving the constraint unchanged.

4.3 Constraint Satisfaction

Constraint Satisfaction refers to the process by which the system ensures that an individual object satisfies all the constraints imposed upon it. The Constraint Satisfaction algorithm may be invoked for many reasons. These include:

- Every time a constraint is activated or deactivated.
- During interactive drags.
- When a feature is adjusted.
- As part of the Constraint Maintenance process.

The process is a relatively simple one, consisting of the following steps:

- Ensure the Origin satisfies any constraints on it.
- Ensure that any alignment constraints are satisfied. The Major axis takes precedence over the Minor axis. If there are only Major constraints active, these are enforced. If there are only Minor constraints active, these are enforced. If both Major and Minor constraints are active, the Major constraints are satisfied, then the Minor constraints are satisfied to the extent that the Major constraints remain satisfied. That means that Minor constraints may not be exactly satisfied, only approximately.
- Ensure that Scaling constraints are satisfied. This can be done without breaking any other constraints.
- If the object is a Torus, ensure that Radius constraints are satisfied. This will maintain the Scaling constraints if any are in place.
- If the object has control points, ensure that each control point satisfies its constraints. The control points are adjusted in order from smallest (0) to largest.

Ensuring that a constraint is satisfied means moving the constrained feature by *the minimum distance* in order to satisfy the constraint. The object changes as if attached to the feature. The Radius feature is a special case. The feature itself will float over the object. The only thing that remains constant (or should) is the distance from the feature (the purple dot) to the center of the torus. Even that isn't strictly true, but is close enough. It is strongly recommended that, if possible, Radius features on the equatorial plane of the torus be used. the equatorial plane is the one passing through the center of the tube, cutting it into a top and bottom half.

Constraint Satisfaction is more intuitive than it sounds. Try it.

4.4 Constraint Maintenance

Constraint Maintenance is the process by which constraints *between* objects are kept consistent. Constraints between objects are introduced by *Reference* points, those selected using button 1 when adding constraints. Reference points move with the object they reference. Hence constraints that use reference points also change with the referenced object. Taking it one step further, the feature constrained by such constraints must now be moved to ensure that it satisfies the changed constraints. Finally, this means that the whole object changes.

In the following discussion, an object is *dependent* on some object if it has a *dependency* on the other object. The *dependents* of a given object are all the objects that are dependent on that object. A *dependency* is introduced by a Reference point appearing in ANY of the constraints on the object.

Dependencies appear in chains, grouping objects together. Object A depends on object B which may depend on objects C and D, and so on. Note also that dependencies imply some order. A dependency is of one object on another. Finally, notice that the chain of dependencies doesn't contain cycles. It is impossible to follow the chain and end up back where you started.

When editing an object, all the dependents of that object are shown dashed. Any dependents of those objects are also shown dashed, and so on. These are all the objects that may be influenced by a change in the current object. Finally, *Constraint Maintenance* is the process of working out which objects are influenced and making the changes.

Constraints are maintained whenever the object is manipulated. It is in fact a final part of the Constraint Satisfaction procedure outlined above. After all the constraints on one object are satisfied, the same process is run on every dependent object in a certain order, ensuring that when all is done every constraint is updated and every object satisfies all its constraints.

Note that the constraint maintenance procedure may take a little while, because it must performs calculations for every dependent object, all the way back along the chain. If it becomes too slow, the process may be turned off for interactive drags by clicking on the *Maintain* toggle in the Edit dialog box.

If you don't want any maintenance, define all position constraints as absolutes, all scaling constraints as offsets and rotation as one or the other. That is, don't use button 1 to select points. Alternatively, deactivate all constraints before you finish editing an object. INACTIVE CONSTRAINTS CANNOT AFFECT THE OBJECT.

4.5 Adjusting Features

By *Adjusting* a feature we mean changing where that feature is relative to the *object*, not the world. So moving the Scaling point to a different vertex is adjusting it. Making the axis point toward the other end of the object is adjusting it. Dragging a feature (and the object) with the mouse is *not* adjusting it.

Adjusting features is almost always required to achieve the desired constraint system. You may need to move the origin to a different place so that you can apply the correct constraints. For almost every object there is some natural choice of features. If you are thinking "I need the bottom of the box to be on top of the other box", then the Origin you want is the bottom of the box. That is, the feature should match your the feature in your thinking (in this case the "bottom").

To adjust a feature, click on the label for its list of constraints (ie the *Origin* label, or the *Scaling* label). In some cases this is a menu, with options for how to adjust the feature. The options are explained below:

Origin: You may choose a single point to become the new Origin, or opt to take the midpoint of 2 points. In each case select the defining points (using the normal procedure) and the Origin will shift.

Scaling: Choose a single point or midpoint of 2 points to become the new scaling point. The psecifying points must (for no good reason) be vertices of the object.

Major and Minor: You may choose a point that the new axis should point toward, or select 2 points that the axis will become parallel to. In the first case the axis changes to point along a line from the Origin toward the selected point. In the latter case it swings to be parallel to the chosen points. Note that the Major axis takes precedence over the Minor axis. In other words the Major axis will always go where you tell it to, but the Minor axis will only change to the extent that the major axis remains unchanged. (If you think about it, that means the Minor axis can only be rotated about the Major axis).

Radius: As for the Origin, you may choose a single point or the midpoint of 2 points.

It doesn't matter which type of point you select. Different point types have no meaning in this context.

When the new feature is selected, the constraints are updated and the constraint satisfaction algorithm run. Hence an object may react quite violently to having its features changed. Be particularly careful with inconsistent constraints which become satisfiable after changing a feature. These may cause very large changes to the object.

4.5.1 Forced Constraints There are some situations in which the relative arrangement of features makes aspects of editing impossible. For example, if the Scaling and Origin points both lie in the same axis plane, it is impossible to scale the object perpendicular to that plane. That is because the distance from the Scaling Point to the Origin Point along that axis is 0. Scaling this distance by anything still gives 0, so the point is theoretically constrained to remain in the plane.

For once there is the opportunity to make theory match reality. The point is theoretically constrained, so we make it practically constrained too. This is easy to achieve by activating the relevant default constraint (there is one default scaling constraint corresponding to each axis). A constraint activated in this way is called a *Forced Constraint*.

Forced Constraints behave in exactly the same manner as any other constraints, but you cannot deactivate them without changing the relative location of the Origin and Scaling points. When the situation leading to the enforcement is changed, the constraint will be deselected automatically (unless you selected it in the interim). So if a constraint appears to activate for no reason, it has probably been forced.

4.6 Interactive Manipulation

If a feature has some freedom in its constraints, it is generally open for interactive manipulation. The easiest way to tell is by moving the cursor over the feature. If the cursor is a small circle, the point CAN NOT be manipulated. If it anything else, it can be manipulated.

The Origin, Scaling and Radius Points are all manipulated by dragging the corresponding point on the screen (using button 1). The Axes are manipulated by dragging in the circle around the object. The interface used is the arcball rotation interface. It should be intuitive (!). Note that because rotation can begin anywhere within the circle, if you accidentally miss the feature you really want to manipulate, the object may rotate. *Undo* gets you back.

When a point is dragged, the exact motion of the point in 3D depends on the constraints imposed upon it. Feedback is provided to indicate which type of constraint is active.

Plane: A large square with a cross through the middle is drawn centered on the feature. Dragging the point drags it around on the plane in 3D.

Line: A line is drawn through the point. Dragging it moves the point along the line.

Sphere: A Circle with 3 arcs is drawn. Motion within the circle moves the point around in the front half of the sphere as seen by you (the half notionally sticking out of the screen). Motion outside the circle is moves the point on the back half of the sphere (the half buried into he screen).

Circle: The constraint circle is drawn on the screen (it will probably be an ellipse, and it's somewhat approximate). The point will move around on the circle.

As a point is interactively dragged, all constraints are maintained (unless explicitly set otherwise).

4.7 The Edit Dialog Box

To begin editing an object, select its name off the *Edit* menu. (Note that if the name of the object has been changed since it was placed on the menu, the menu label will still use the old name.) When an editing session begins on an object the Edit dialog box is popped up below the window in which the editing is happening (you can move the dialog around at will, even minimize it).

Some of the buttons in the Scene and CSG windows will become inactive during editing. Those still active will work as they normally do.

It is sort of possible to edit more than one object at a time. Selecting another object off the *Edit* menu while already editing an object will cause editing to switch to the new object. Then, when that object is finished, editing switched back to the original object. It is really an edit stack — you edit the object on top of the stack.

The *Edit Dialog* box contains a row of buttons and a group of constraint lists. There is one list of constraints for each feature. The preceding sections describe how to use and modify these lists. The buttons available are:

— *Finish:* This finishes a session. Several things are done:

- The object is removed from the edit list.
- All objects depending on this one have their constraints updated and reworked.
- The screen is redrawn and all the main buttons resensitized.
- If another object is on the stack, it is edited.

There is no return from a finish. An full undo should to be provided, but isn't.

— *Suspend:* This is the same as a finish, except that the object is not removed from the edit list. Use this it you intend to do more editing of the object in the near future.

— *Undo:* Undo provides undo right back to the start of the current session. Everything can be undone, including constraint addition, removal, selection and all drags of any type. In effect, undo winds the state back one step. Undo is also used to cancel point selection, cancel a view change and cancel constraint modification.

— *Redo:* Undoes an Undo.

— *Maintain:*

The Maintain button is a toggle. It controls interactive constraint maintenance. If on, constraints will be interactively maintained as objects are dragged and edited. If off, constraints will only be maintained at the end of each object edit session. On is the better setting from almost all points of view, except refresh speed. Turn maintenance off if the number of dependent objects is large and updating is much slower than usual.

4.8 Shortcuts

There are shortcuts available for adding constraints, adjusting features, and executing general editing commands. All commands accessed in this way behave identically to the corresponding command accessed via the edit dialog box.

Clicking with button 2 over a feature pops up a dialog that emulates the menu found in the edit dialog box under each feature label. This provides access to the feature adjustment facilites. Note that the window that pops up is actually a set of command buttons, even though it looks like a menu. This is, as far as I can tell, some sort of limitation with X (I would have preferred menus).

Clicking with button 3 over a feature pops up an Add Constraint type window, from which you can select a constraint type to add. This emulates the behaviour of the Add menus in the edit dialog.

A button 2 or 3 click that is not over a feature pops up a window containing the general edit commands - Finish, Suspend, Undo and Redo.

4.9 Triangles

The control points of a triangle are attached to each corner of the triangle. They move independently of everything else, but are affected by other operations. That is, they will translate, scale and rotate with the triangle, unless otherwise constrained. It's easiest to play with it and work out what happens.

When constraining control points, control points may only depend on other control points of lower number. So control point 0 cannot depend on any other control point. Control point 1 can depend on control point 0, but not control point 2, and so on.

4.10 Bezier Patches

It may prove useful to read about bezier patches in a graphics book. In brief, they are an approximation surface. By approximation we mean that they approximately follow their control points, but not exactly. What is certain is that the patch passes through the 4 corner control vertices (0, 1, 2 and 3). Also, the surface is tangential to the lines joining the corners to the control points one point further in.

As with triangle control points, bezier patch control points may be dependent on lower numbered controls. The control points are ordered in a way that makes them most useful for this purpose (the ordering may seem strange otherwise).

To join two patches smoothly (with tangential continuity), constrain the edge control vertices for the two patches to be identical, and the control points one in from the edge to be colinear on either side of the boundary.

The control vertices will generally move with the patch as it is translated, scaled or rotated, unless constrained to do otherwise.

4.11 Hints

It is difficult to say beforehand how little or much an object should be constrained. The more it is constrained the better control you have over it under the influence of other editing. However, it takes time to constrain an object and there may be no benefit to doing so (particularly if it has no natural dependencies on other objects). The moral of the story seems to be control a lot or a little, but not in between. Also, avoid loosely specifying any feature, particularly the Scaling Point. Very strange things tend to happen when the constraint maintenance process is run on a partly constrained Scaling Pt.

It is more accurate to use separate plane and sphere constraints to get a circle constraint. It's not entirely clear why, but in practice it appears to work better.

Ratio points and planes are very useful for solving the age old problem with CSG difference operators, where you wish to avoid ghost surfaces at the subtraction boundary. Ratio constraints with external ratios can be used to constrain the subtracted surface a little way out from the main surface, avoiding the ghosting.

5. THE CSG INTERFACE

The CSG interface consists of a separate window in which CSG objects are created and edited. The window is accessed via the *CSG Window* button on the main display. The window is divided much along the same lines as the scene window, with the addition of an extra region below the View Window of a *CSG Tree Window* where the objects currently being manipulated are represented by small labeled buttons. Each button is a menu, allowing access to certain functions for that object. The available functions will be discussed later.

The relative size of the CSG Tree Window and CSG View Window can be changed by dragging the small square on the dividing line up or down. Scrollbars will appear in both windows if required.

5.1 CSG Window Functions

Many of the functions provided in the CSG window are identical to those in the Scene Window. The full list of functions available is:

- **CSG:** Modify Existing, Copy Existing, Save OFF, Delete Existing, Close
- **Object:** New, Edit, Name, Attribs, Alias, Dense Wire, Thin Wire, Change Base
- **View:** Viewpoint, Pan, Lookat, Lookup, Distance, Eye
- **Window:** Zoom, View Size, Draw Mode, Save, Recall, Delete
- **Layers:** New, Add Objects, Merge, Display
- **Clear CSG**

In addition, there is an Edit button as with the Scene Window.

For those functions which are also defined in the Scene Window, the behaviour is the same except that it works on the CSG View Window and objects contained therein. One exception is the New Object dialog. It will not allow you to select triangles, planes or patches. These objects to not enclose space, and hence are unsuitable for CSG operations.

A word on attributes is required at this point. By default CSG objects have no attributes attached. Hence attributes attached to an instance of a CSG object will be used. However, if attributes are attached to the components of a CSG object, those attributes will take precedence over attributes attached to the instance. So if you specify a window, and attach attributes to the glass but not the frame, you can specify a different frame color for each instance through the instance attributes, but the glass will always be glass. In other words the CSG objects inherit attributes from the instance, unless they themselves have attributes attached. In another set of words, attributes are percolated from the top down, with component objects getting the first set of attributes found on the path from the object to the root instance.

5.2 Modify Existing

This function allows for the editing of an existing CSG type, which is selected through the general object selection dialog (like a new object). There are some restrictions on the use of this function. If a CSG object has instances that appear in other CSG objects, then that object cannot be modified. If there are any other instances (that appear in the scene), they will be highlighted to indicate their presence. You then have the option of marking these instances for later modification, or copying the CSG object rather than modifying it. If the objects are marked for modification, a *Change Base* operation will be performed on them when the modified object is completed. If a copy of the CSG object is edited than no changes will be made to existing objects.

The CSG object that is being modified is stored with the tree during editing. If a modified tree is combined with some other tree, the resulting tree is considered to be the modified tree for the purpose of later *Change Base* operations on instances.

This button will be unavailable if there are no CSG objects currently defined.

5.3 Copy Existing

The CSG object selected is copied, and the copy placed in the *CSG Tree Window* for editing. Beyond this point the tree behaves like any other.

This button will be unavailable if there are no CSG objects currently defined.

5.4 Save OFF

The *Save OFF* function allows a CSG object to be saved as an OFF format file. The object selection dialog is presented, allowing you to choose the object to save. The file selection dialog is then presented, allowing you to specify the filename. The default filename provided is the name of the object suffixed with .aoff. The file name you specify should end with .aoff. The other files saved as part of the object will be suffixed accordingly.

5.5 Delete Existing

A dialog is presented allowing the choice of an existing object to be deleted. The object to be deleted cannot have any instances.

This function will be unavailable if no CSG objects are defined.

5.6 Close

This button closes the CSG Window. It only removes the window from the screen, it does not change the contents of the window, which will be available once the window is remapped using the CSG Window button in the Scene Window.

5.7 Tree Menu Functions

Each button displayed in the CSG Tree Window is a menu displaying functions applicable to that particular node in the tree. Which functions are applicable depends on the type of node and where it is in the tree. The full list of available functions is:

- *Display / Hide*
- *Move*
- *Attach*

- *Complete*
- *Preview*
- *Evaluate*
- *Reorder*
- *Break*
- *Copy*
- *Delete*

5.8 Display or Hide

This option toggles display of the tree in the CSG View Window. The objects in a displayed tree may be edited just as normal instances. There are a few things to note however. All that really matters is the position of the object relative to other objects IN THE SAME TREE. Once a CSG object is created it can be moved about just as any other object, so its position with respect to the origin is not important for that reason. However, it does matter where the object is relative to the origin, because the origin will become the notional center for the created CSG object. That is, it will be the default object origin for position, and the default fixed point for scaling and editing. These points may be changed later, but it is still a good idea to keep the origin somewhere near where it is sensible to think of it as the center. Also note that the size of the object is not too important, as the resulting object may be scaled later.

Removing an object from the display takes any of its instances off all of the lists, including the selection and edit lists. In other words it is only possible to operate on displayed objects.

Displayed objects are still affected by layer visibility, so it they don't appear, check the display of layers.

The Display option is available for root nodes. All or none of a tree is displayed. To display parts, break the tree then put it back together when finished. The label on this menu item changes to indicate whether or not the tree is currently displayed.

5.9 Move

The move option is for reordering trees within the CSG Tree Window, NOT for moving a tree's instances in the world. This option mostly exists because it is possible for some objects to be off-screen, while attachment requires that the 2 objects to be attached both be on-screen. Off-screen objects can be accessed using the scroll bars, but it may not be possible to get both trees on the screen without moving one.

Once the Move option is selected, a small rectangle appears attached to the cursor. Click with button one when the cursor is in the position that you want the tree to appear. For instance, to move the 7th tree so that it appears after the 2nd tree, select move from the 7th tree's menu, then click with button 1 somewhere between the 2nd and 3rd trees.

The Move option is available for root nodes.

5.10 Attach

This operation is the essence of CSG, allowing 2 trees to be combined with a set operator.

To attach one tree to another, select the Attach item, then choose the node from ANOTHER tree that you wish to be the SIBLING of the attached tree. The node selected as the sibling must be from a different tree - you cannot attach a tree to one of its children. The node you attach will become the RIGHT child of a new node, and the node you selected as the destination will become the LEFT child of the new node. The new node will be attached to the parent of the destination node, or become a new tree if the destination was a root node itself.

To choose the destination, push button 1 down on the desired destination node, and another menu will pop up allowing you to choose the type of operation to apply, or cancel. Choose the appropriate option and release the button.

Note that the ordering of the nodes is only really important for the difference operator. The right node is taken away from the left node. Use the Reorder option to swap the ordering if it isn't right.

Regardless of the initial display state of the tree you are attaching, it will always assume the display state of the destination tree.

This option is available for root nodes.

5.11 Complete

The Complete option takes a CSG tree and turns it into a new CSG object. If the tree selected consisted of a single instance it will simply become a normal instance in the world. Otherwise the following things happen:

- All references and dependencies to objects NOT in the current tree are replaced by absolute (for Origin) or offset points.
- You are prompted for a name for the CSG object being created. Select Complete or hit return when you are finished, or cancel at this point.
- A wireframe is generated for the object. THIS MAY TAKE A WHILE. It could be as bad as O squared seconds, where O is the number of objects in the CSG tree you are creating. (The process actually is most dependent on the number of polygons making up the objects at each level in the tree, but this is roughly proportional to the number of objects.) Dense wireframes significantly slow the process, but give superior results. While the wireframe is being created you will get occasional updates as to its progress. Click on Cancel to cancel wireframe generation. Note that it might take a while to respond to a cancel message.
- You are presented with the resulting object and asked to specify a default Scaling Point for the object. You may still cancel at this point.
- The object is made available as a new CSG object.

- If the tree was a modified tree (it originated from a Modify Existing call), instances of the object will be adjusted to match the new object.
- The tree is removed from the CSG Window.

The Complete option is available for all root nodes.

Note that no instances of the object are created, you must create instances through the New Object dialog, just as with any other object.

The wireframe generation procedure takes a while because it attempts to produce a realistic wireframe for the object which is not more complex than necessary. This procedure is also highly susceptible to numerical inaccuracy. Occasionally it produces wireframes which aren't quite right.

When asked to select a default Scaling Point, the option is also available to use a full wireframe for the object. The full wireframe consists of the union of all component object wireframes. This option is useful in cases where the intersecting set of objects is so small that the approximate wireframes do not intersect at all. It may also be useful if the wireframe generation procedure got it horribly wrong.

5.12 Evaluate

The Evaluate option is combined Complete and Attach operation. The subtree starting at the node selected is completed as above, then an instance of this completed object is attached in place of the subtree. It allows for tree simplification to some extent.

When you select this option, the processes involved are the same as for complete with the extra stage at the end of creation and attachment of an instance in place of the tree.

This option is available for non-root internal nodes in the tree.

5.13 Preview

The subtree of the selected node is previewed using a renderer you choose. The options for the preview are the same as those for *Preview* in the *Scene Window*. A single instance of the subtree is created, along with a default light, and these are exported to the renderer of choice for preview.

5.14 ReOrder

When this option is selected the children of the selected node will be reordered. This is useful for Difference operators, where the ordering matters (right is removed from left), or for changing the appearance of a tree in the window.

This option is available for all internal nodes.

5.15 Break

Selecting Break will cause the selected subtree to be broken off its tree and made a tree by itself. The parent node of that selected will be removed, and the node's sibling put in the parent's place. The node and all its children will become a separate tree.

The Break option is available for all non-root nodes.

5.16 Copy

The copy button copies the selected node and all its children and adds them as a new tree. It replaces the Copy command on the Object menu. The new tree is not initially displayed.

All constraints in a copied tree that reference other nodes in the tree will be adjusted to reference the corresponding copies. This allows a completely constrained tree to be copied and then edited independently.

Any node may be copied.

5.17 Delete

This will delete the selected node and all its children, patching the remaining tree as appropriate. No confirmation is asked for, so be careful. This function is essentially a replacement for Delete on the Object menu.

Any tree can be deleted.

6. COMMAND LINE OPTIONS

-F filename

Load the specified filename on startup. The -F can be omitted. So *sced -F file.scn* is the same as *sced file.scn*. The file is searched for in the current directory first. If not found, and the name does not end in .Z or .gz, one of these options is added and the name is tried again. The extension added depends on whether gzip exists on your system at compile time. If the file is still not found, the procedure is repeated but in the default scene directory. If still not found, it is assumed the file does not exist, and the name is used as the default name for saving.

-D defaults

Use default values found in the given file. If this option is not present the program will look for a file called .scenerc in the user's home directory. Failing this, internal defaults will be used. The format for the defaults file is described below.

-I WidthxHeight

Use a view window of the given size. This does not effect the size of the program's window (use -geom), rather it determines the size of the scrollable window in which the scene appears.

7. DEFAULTS FILE

At startup the program attempts to read a file containing default values for various parameters. If the -D option was specified on the command line, it will look there, otherwise it will look for a file called *.scenerc* in your home directory. If neither can be found, compile time defaults will be used.

7.1 Format

The following keywords with the accompanying arguments are accepted. If a keyword is not present in the file, that value will have the built in default. Spaces, tabs and newlines act as separators. Anything from a # to the end of line is taken as a comment. In the following list, keywords are in bold and arguments in italics. Note that the program is case sensitive. If a string is required, it must be delimited by double quotes ("). A double quote may be contained within a string by preceding it with a backslash. A newline may be inserted using backslash n (C syntax).

MainViewport or **CSGViewport**
: Begins the definition of the main or csg viewing parameters. Some or all of the following keywords should appear next in the defaults file.

LookFrom *x y z*
: Set the *Viewpoint* vector to given vector, where x, y, and z are floating point numbers.

LookAt *x y z*
: Set the *Look At* point to be the given point.

LookUp *x y z*
: Set the *Look Up* vector the the given vector.

ViewDist *dist*
: Set the viewplane *Distance* to the given value, which should be a positive value.

EyeDist *dist*
: Set the *Eye Distance* to the given value, which should be positive.

Magnify *zoom*
: Set the *Zoom* value to the given value, which should be a positive integer.

Screen *width height*
: Set the *Image Size* to the given width and height, which should be positive integers.

Viewport *"name"*
: Define a viewport which will be available for recall. The parameters that follow are all the same as for the MainViewport or CSGViewport. Useful default viewports include plan and elevation views. *Name* is the name of the label to use in the view dialog boxes.

POVray *path options version*

Rayshade *path options*

Radiance *path options*

Renderman *path options*

VRML *path options*

Genray *path options*

Genscan *path options*
Path and options are strings. Set the pathname and options for the named renderer. These values will be used for the preview command - they do not influence the exported scene. If you do not have one of renderers, set the path string to "", the empty string. POV has the additional argument of a version number. This should be 3 for POV version 3.0, or 2.2 for earlier versions. Note that earlier versions of POV (version 2.2) must be patched.

Directory *pathname*
Set the default directory for loading, saving and exporting scenes. This is simply the directory that is shown in the file selection dialog box, and is not any hard restriction.

Attributes
Set the default object attributes. Any or all or the following options can appear. If a particular option is not set, the value will retain the compiled in default. See the *Attributes* command above for a full description of each parameter.

Color *red green blue* or **Colour** *red green blue*
Sets the default object color, where red, green and blue are values between 0 and 1.

Diffuse *coef*
Set the default diffuse coefficient.

Specular *coef power*
Set the default specular coefficient and power.

Reflect *coef*
Set the default reflective coefficient.

Refract *coef*
Set the default coefficient of refraction.

Transparency *coef*
Set the default transparency value.

Autosave *minutes*
The number of minutes between automatic saves. If 0, autosaving is turned off.

Compress
Compress files on output. All files saved will be piped through gzip, or compress if gzip is unavailable. The filename will have a .gz (or .Z) appended if not already present. Sced knows how to load compressed files (which it recognizes through a .gz or

.Z extension). This is intended to save space for files with large numbers of CSG objects (and hence wireframes).

Wireframe Full

Save full wireframes. By default, Sced only saves the basic version of CSG wireframes, which is the wireframe before simplification for display. It then repeats the simplification process when the wireframe is loaded. To save both the basic and simplified wireframe, use this option in the defaults file. Saving full wireframes will take more space, but speed the loading of files.

Target *renderer*

The named renderer (one of Rayshade, POVray, Radiance, Renderman, Genray, or Genscan) will be used as the default target. Use this option if you export almost exclusively to one renderer.

Declare *renderer* declarations

Renderer is one of POVray, Radiance, Rayshade, RenderMan or VRML (case matters). Declarations is a string. The string is taken as the default declarations for that renderer, to be exported at the start of any file. The string may be modified after startup through the *Declarations* button in the *Target Specific* attributes dialog.

8. X RESOURCES

There are fallback resources for all the top level window sizes. Everything else uses the local default. If you wish to change things, particularly the font, colors and geometries you may do so through a resources file, the command line or compile them in. The font has a major influence on the relative size of buttons and some dialogs. You may prefer editing using white on black, and if you have a smaller screen resolution you will most likely wish to change the default geometries. The geometries used by default were designed for at least 1024x768, but are best at bigger than 1152x900 screen resolution.

The widgets geometries you may like to change, and their defaults are:

Sced.geometry: 800x600 — the main window overall size.

Sced.csgShell.geometry: 800x600 — the csg edit window overall size.

Sced.newObject.geometry: 400x300 — the new object selection box size.

Sced.csgSelectShell: 400x300 — the dialog for selecting CSG objects for new object or any of the existing CSG object operations.

Sced.wireSelectShell: 400x300 — the dialog for selecting Wireframe objects for new object or wireframe delete.

Sced.csgReferenceShell: 400x400 — the "Choose a Scaling Point" dialog associated with CSG completion.

Sced*mainViewWindow — the main view window. You may wish to change the "foreground" and "background" resources for this widget.

Sced*csgViewWindow — the csg view window. Again, you may wish to set the "foreground" or "background" resource.

Sced*csgReferenceView — the view window used to show the "Choose a Scaling Point" CSG completed wireframe. You might like to change the foreground and background.

A large number of other resources are also defined, mostly for controlling colors and sizes. To specify a color, use its name from the rgb.txt database. Defaults are shown in brackets. The resources are:

Sced*font — if you want to change the font.

Sced.xAxisColor — the color of the world X axis. (red)

Sced.yAxisColor — the color of the world Y axis. (green)

Sced.zAxisColor — the color of the world Z axis. (blue)

Sced.axisWidth — the line width for the world axis lines. (2)

Sced.xAxisLength — the length of the X axis line (from the origin to the tip). This value (an integer), is divided by the axisDenom resource to allow for fractional values. Note that lengths are specified in world co-ordinates, not screen size. (2)

Sced.yAxisLength — the length of the Y axis line (from the origin to the tip). This value (an integer), is divided by the axisDenom resource to allow for fractional values. (2)

Sced.zAxisLength — the length of the Z axis line (from the origin to the tip). This value (an integer), is divided by the axisDenom resource to allow for fractional values. (2)

Sced.axisDenom — the value by which each world axis length is divided to get the true length. (1)

Sced.axisFont — the font for the letters that label the axes. (default font)

Sced.majorAxisColor — the color of the Major Axis for editing. (red)

Sced.minorAxisColor — the color of the Minor Axis for editing. (green)

Sced.otherAxisColor — Not used.

Sced.editAxisWidth — the line width for the edit axes. (2)

Sced.majorAxisLength — the length of the Major Axis line. This value (an integer), is divided by the editAxisDenom resource to allow for fractional values. Note that lengths are specified in world co-ordinates, not screen size. (4)

Sced.minorAxisLength — the length of the Minor Axis line. This value (an integer), is divided by the editAxisDenom resource to allow for fractional values. Note that lengths are specified in world co-ordinates, not screen size. (3)

Sced.otherAxisLength — Not used. (2)

Sced.editAxisDenom — the value by which each edit axis line length is divided to get the true length. (2)

Sced.editPointRadius — the radius of the circle drawn around the Origin and Scaling edit points. This value also sets the effective size of these points for mouse interaction. (10)

Sced.controlPointRadius — the radius of the circle drawn around bezier and triangle control points. This value also sets the effective size of these points for mouse interaction. (10)

Sced.scalingColor — the color of the edit Scaling Point. (red)

Sced.originColor — the color of the edit Origin Point. (green)

Sced.torusRadColor — the color of the edit Torus Point. (purple)

Sced.controlColor — the color of bezier and triangle control points. (green)

Sced.controlTextColor — the color of the text labelling bezier and triangle control points. (DarkGreen)

Sced.controlFont — the font used to label the control vertices. (default font)

Sced.objectColor — the color of an object that is being edited. (blue)

Sced.selectColor — the color of selected objects. (red)

Sced.selectWidth — the line width of selected objects. (2)

Sced.lightColor — the color of lights on screen. (yellow)

Sced.cameraColor — the color of camera on screen. (midnightblue)

Sced.constructColor — the color of construction objects on screen. (green4)

Sced.aliasColor — The color of aliased objects on screen. (maroon)

Sced.lightPointRadius — the radius of the circle drawn around lights on screen. (12)

Sced.constraintColor — the color of constraint indicators, such as lines or planes. (grey)

Sced.planeConLength — the edge length, in world co-ordinates, of the square drawn to represent plane constraints. (6)

Sced.lineConLength — the length, in in world co-ordinates, of the line drawn to represent line constraints. (6)

Sced.pointConWidth — the radius of the circles drawn to represent point constraints. (15)

Sced.inconConLength — the size of the crosses used to represent inconsistent constraints. (15)

Sced.originConWidth — the size of the squares drawn to indicate points specifying origin constraints. (7)

Sced.scaleConWidth — the size of the squares drawn to indicate points specifying scaling constraints. (5)

Sced.rotateConWidth — the size of the squares drawn to indicate points specifying alignment constraints. (3)

Sced.torusRadConWidth — the size of the squares drawn to indicate points specifying torus radius constraints. (9)

Sced.controlConWidth — the size of the squares drawn to indicate points specifying control point constraints. (11)

Sced.referencedColor — the color that referenced objects are drawn during point selection. (red)

Sced.activeColor — the color of the currently active point for point selection (the point that will be selected). (red)

Sced.selectPointColor — the color of points already selected. (blue)

Sced.selectPointWidth — the width of the square drawn around the active and selected points. (8)

Sced.selectPointLineWidth — the width of the line used to draw the above square. (2)

Sced.absoluteColor — the color used to represent absolute point constraint specifiers. (green)

Sced.offsetColor — the color used to represent offset point constraint specifiers. (blue)

Sced.referenceColor — the color used to represent reference point constraint specifiers. (red)

Sced.arcballColor — the color used to draw the arcball arc. (grey)

Sced.editOrientation — the preferred orientation of the boxes within the Edit window. Legal values are *horizontal* or *vertical*. If horizontal, constraint boxes are stacked in rows, with a new row started only if required. If vertical, boxes are stacked in columns, with a new row only started if required. Use horizontal if you want a short wide edit box. Use vertical if you want a tall narrow dialog. (horizontal)

Sced.editRows — Only applicable if editOrientation is vertical, this sets the number of boxes in any one column. (5)

Sced.editColumns — Only applicable if editOrientation is horizontal, this sets the number of boxes in one row before the next is started. (5)

9. EXTERNAL FILE FORMAT

Sced supports an external file format that is easy to read and create by hand or through external programs. Any file not recognized by Sced as its own internal format will be processed based on the assumption that it is an external file.

Anything following a #, and up to the end of the line, is a comment and will be ignored.

The key words and their arguments are described below.

Version *version_num*: Specifies the Sced version number that this file applies to. At the moment the only valid value for *version_num* is 0.8. This command must be the first in the file. This option is included primarily to ease the transition to later versions.

Viewport: Begin definition of the viewport. Parameters that are specified will modify the main view in place when the file is loaded. Definition of the viewport ends when a command not included in the parameter list below is encountered. Valid parameters to follow this keyword are:

- **LookFrom** *x y z*: Set the view direction such that you appear to be looking down *x y z* toward the *LookAt* point.
- **LookAt** *x y z*: Set the point to look at. This point will appear in the center of the view window.
- **LookUp** *x y z*: The vector *x y z* will always appear to be pointing vertically up in the window.
- **Viewdist** *d*: Set the distance of the viewing plane from the *LookAt* point.
- **EyeDist** *d*: Set the distance of the eye from the viewplane. This controls perspective effects, and the field of view.
- **Magnify** *s*: Set the magnification of the image. This is equivalent to the *Zoom* interface function.
- **Screen** *x y*: Set the image size.

Camera: Begin definition of the camera. The format is identical to the definition of the *Viewport*, but modifies the world camera instead.

Object *name base*: Begin definition of an object instance. It has name given by *name* and is of base type *base*. *Name* and *base* must be strings enclosed in double quotes. *Base* must have been previously defined. The default bases are: *sphere*, *cube*, *cone*, *cylinder*, *square*, *plane*, *light*, *spotlight* and *arealight*. The instance is modified by the following parameters. Definition ends when a parameter not listed below is encountered.

- **Matrix** *x1 x2 x3 y1 y2 y3 z1 z2 z3*: The object is transformed by the given matrix. *x1 x2 x3* is the first row, *y** the second row and *z** the third row. The matrix is pre-multiplied onto the current transformation matrix (which starts as the identity).

Scale *x y z*: Scale the object by *x* in the X direction, *y* in the Y direction and *z* in the Z direction. The scaling is performed in the order it is given.

Rotate *x y z*: Rotate the object through *x* degrees about the X axis, followed by *y* degrees about the y axis, followed by *z* degrees about the Z axis. Note that the order of rotation matters. Separate rotation statements are applied in the order given.

Position *x y z*: Translate the object along the vect *x y z*. Translations accumulate, and are applied after all other transformations.

Dense *level*: Set the wireframe density level of the object to *level*, which should be an integer. The default level is 0. Each extra level doubles the number of generators for the object, or multiplies by 4 the number of facets on a sphere.

Color *r g b* or **Colour** *r g b*: Set the object's color, or intensity if it's a light source. *R g b* should be values between 0 and 1.

Diffuse *v*: Set the diffuse surface reflectance to *v*.

Specular *v p*: Set the specular surface reflectance to *v* with an associated power of *p*.

Reflect *v*: Set the reflectance of the surface to *v*.

Refract *v*: Set the index of refraction of the body to *v*.

Transparency *v*: Set the transparency of the object to *v*.

Extend *string*; Set the extension string for the object.

CSG *name*: Begin definition of a CSG base object which will have the name *name* for later reference. *Name* must be a string enclosed in double quotes. Following *name* should appear a valid CSG object tree description, as defined here. A CSG tree definition is best described recursively:

- A CSG tree may be one of the words *Union*, *Intersection* or *Difference*, immediately followed by two CSG tree definitions.
- A CSG tree may also be a single *Object* command describing a leaf of the CSG tree.

For example,

```
CSG "example"
Union
        Difference
                Object "leaf1" "cube"
                Object "leaf2" "sphere"
        Object "leaf3" "cylinder"
```

describes a CSG object consisting of the difference of a cube and sphere unioned with a cylinder.

Wireframe *name num_verts num_faces verts faces*: Begin description of a new wireframe type object. The object has name *name* (a string). It has *num_verts* vertices and *num_faces* faces. *Verts* is a list of vertices, each given as an *x y z* vector. *Faces* is a list of faces, each given as a number of vertices followed by the indices of the vertices in the vertex list. Indices start at 0.

Each face must have at least 3 vertices, and be convex. Face vertices should be given in a clockwise order as viewed from outside the object. If you intend to use the wireframe in a CSG object, the faces must intersect each other only along complete edges or at a single vertice. The object must also be closed.

For example,

```
Wireframe "mycube" 8 6
1 1 1
1 -1 1
-1 -1 1
-1 1 1
1 1 -1
1 -1 -1
-1 -1 -1
-1 1 -1
4 0 3 2 1
4 4 5 6 7
4 0 4 7 3
4 1 2 6 5
4 0 1 5 4
4 2 3 7 6
```

defines a cube.

10. GENERAL NOTES

If Sced aborts for some unrecoverable error the scene at the time of the error is dumped to a file called sced.save in the current directory. Unrecoverable errors include running out of memory and errors in the CSG wireframing code.

The file selection code traverses the whole filesystem at times, which is very annoying. The really annoying thing is that it sometimes hangs on bad NFS links. If that means anything to you then it may be useful, otherwise forget it. It also means that sometimes the program ends up in the root directory. I don't know why. It's really, really annoying. If have have a better file selection dialog box I'd like to hear from you.

11. RENDERER SPECIFIC NOTES

11.1 POVRay

If you are not using POV 3.0, the patches supplied with the Sced distribution will need to be applied. They allow for the matrix keyword to specify object transformations.

Wireframes are exported as the union of their component polygons.

Sced defines the identifier *ambient_light_level*, which holds a float value equal to the ambient light level in the scene, as entered in the *Ambient* dialog. You may use this identifier in texture statements to save adjusting all the attributes every time you change the ambient light value.

The Specific Attribute string is exported after the geometry, either before or after the transformation depending on the *Transform Texture* toggle. It should be a complete texture string. Take particular care to balance brackets. /P Directional light sources are exported as point sources.

11.2 Rayshade

The simple attributes on an object are exported even if more specific attributes are given. This allows the surface properties to still be applied. The problem arises out of Rayshade's strange way of entering attributes in two separate groups, surfaces before the object definition and transformation, and textures afterward.

Wireframes are exported as triangles, and phong triangles if vertex normals were specified in the OFF file that defined the wireframe. The triangulation method used is trivial, and will fail for concave polygons.

Rayshade does not have a surface patch primitive. Bezier patches are exported as phong (smooth) shaded triangles, of the density used in the scene.

11.3 Radiance

Radiance differs significantly from the other renderers, so these notes are extensive.

Point light sources are exported as small spheres. There is no need to limit the intensity of a light source to 1.0, you may go as high as you wish. Directional light sources are also exported in this manor.

Radiance only allows for true spheres, cones and cylinders. This precludes a very large range of transformations (it only allows for uniform scaling in the radial direction). To overcome this difficulty, Sced exports any non-true object as a list of polygons. Obviously this impacts on the quality of the picture. You should make extensive use of the *Dense Wireframe* command to improve the appearance of these objects. They will be exported at the same density as they appear on screen.

There is no Torus primitive in Radiance. Tori are exported as polygonal objects. Once again you should make extensive use of the Dense Wireframe command.

The simple surface attributes are exported as a plastic surface, so many of the parameters have no effect. To get anything but the most simple effects you will need to make extensive use of your own modifiers and renderer specific definitions. It also seems that reflection is best modeled through the specular field. So use Specular rather then Reflection to get mirror like surfaces.

Radiance does not have CSG definition within its input language, and instancing is prohibited because of the limited range of transformations available. Hence Sced exports all CSG object instances as individual lists of polygons. This has two effects. Firstly, the exported file becomes very large. Secondly, the representation is only as accurate as the wireframe displayed on screen by Sced. Once again you should make extensive use of the *Dense Wireframe* functions to improve the appearance of CSG objects.

Planes are exported to Radiance as a 10x10 square. Relating this to the plane representation used by Sced, the polygon is the same size as the extremities of the cross drawn.

When you export a radiance file, an extra dialog is used to enter parameters like the interior/exterior option, the view volume and assorted other things. These options are exported to a rad (.rif) input file. The geometry and modifiers are exported to a .rad file, ready for use with oconv. I recommend setting your previewer to "rad" with options "-o x11" to get an interactive preview using rview.

11.4 RenderMan

RenderMan is an excellent language to export to, not surprisingly.

The color of an object is always exported. So you should set the red, green and blue values in the simple attributes box every time you want attributes, even if those attributes are target specific.

Planes are exported as 10x10 squares, as with Radiance.

Polygons are exported as the simple polygon type, which doesn't support concave polygons. I couldn't get the other method (using GeneralPolygon), to work.

When exporting, you are prompted for the name of the image file to render. This defaults to the base filename with .tif added.

11.5 VRML

By convention, VRML worlds have Y as the up vector. With this in mind, you might like to set the default view to have Y as up.

Surface patches are exported as triangle sets, with vertex normals defined.

12. ACKNOWLEDGEMENTS

Thanks to all the people who helped out with comments and suggestions and debugging. If I start mentioning names I'll forget someone. Sced would not be as good as it is without the help of others.

APPENDIX A: Text Editing Commands

The following list of commands is taken from a description of the Athena Text widget. Note that the all the text widgets in Sced (anything you type text into) support these functions. You may also select from and copy into these text windows, just as you would an xterm (button 1 select, button 2 paste, button 3 continue selection).

The commands are:
Ctrl-A: Beginning of line.
Ctrl-B: Backward character.
Ctrl-D: Delete next character.
Ctrl-E: End of line.
Ctrl-F: Forward character.
Ctrl-G: Reset the multiplier.
Ctrl-H: Delete previous character.
Ctrl-J: Newline and indent.
Ctrl-K: Delete to end of line, putting it in buffer 2.
Ctrl-L: Redraw display.
Ctrl-M: Newline.
Ctrl-N: Next line.
Ctrl-O: Newline and back up.
Ctrl-P: Previous line.
Ctrl-R: Search backward (a dialog is popped up asking for the string to search for).
Ctrl-S: Search forward.
Ctrl-T: Transpose characters. The characters on either side of the cursor are swapped.
Ctrl-U: Multiply the next action by 4.
Ctrl-V: Next page.
Ctrl-W: Delete the selection and put it in buffer 2 (not the selection buffer).
Ctrl-Y: Insert the contents of buffer 2.
Ctrl-Z: Scroll up one line.
Alt-B: Backward word.
Alt-F: Forward word.
Alt-I: Insert file (a pop up dialog requests the name).
Alt-K: Delete to end of paragraph, stuffing it in buffer 2.
Alt-Q: Form paragraph (this formats a paragraph).
Alt-V: Previous page.
Alt-Y: Insert the selection.
Alt-Z: Scroll down one line.
Alt-d: (case sensitive) Delete next word.
Alt-D: (case sensitive) Delete next word, stuffing it in buffer 2.
Alt-Delete:
Alt-Backspace:
Alt-h: (case sensitive) Delete previous word.
Shift-Alt-Delete:
Shift-Alt-Backspace:
Alt-H: (case sensitive) Delete previous word, stuffing it in buffer 2.
Alt-<: Beginning of file.
Alt->: End of file.
Alt-]: Forward paragraph.
Alt-[: Back paragraph.

APPENDIX B: Special Translations

Shift Drag: Change viewpoint.
Alt Drag:
Meta Drag: Pan.
Ctrl Drag: Select and Edit.
+: Zoom double.
-: Zoom half.

All except Ctrl Drag work at all times, including during point selection and object editing.

CONTENTS